# Ontology-Mediated Queries Distributing over Components

# Ontology-Mediated Queries Distributing Over Components

**Gerald Berger** **Andreas Pieris**

Institute of Information Systems, TU Wien, Austria

{gberger,pieris}@dbai.tuwien.ac.at

## Abstract

Ontology-based data access is concerned with the problem of querying incomplete data sources in the presence of an ontology. A key notion in this setting is that of ontology-mediated query, which is a database query coupled with an ontology. An interesting issue is whether the answer to an ontology-mediated query can be computed by parallelizing it over the connected components of the database, i.e., whether the query distributes over components. This allows us to evaluate the query in a distributed and coordination-free manner. We investigate distribution over components for classes of ontology-mediated queries where the database query is a conjunctive query and the ontology is formulated using existential rules. For each such class, we syntactically characterize its fragment that distributes over components, and we study the problem of deciding whether a query distributes over components.

## 1 Introduction

*Ontology-based data access* (OBDA) has recently emerged as a promising application of knowledge representation and reasoning technologies in information management systems. The aim of OBDA is to facilitate access to data that is significantly heterogeneous and incomplete. This is achieved via an ontology that provides a unified conceptual view of the data, and makes it accessible via database queries formulated solely in the vocabulary of the ontology. The actual database query and the ontology can be seen as two components of one composite query, called *ontology-mediated query* [Bienvenu *et al.*, 2014]. OBDA can then be realized as the problem of answering ontology-mediated queries.

A challenging and important topic for declarative database query languages, and in particular (extensions of) Datalog, is coordination-free evaluation. This has its roots in *declarative networking* [Loo *et al.*, 2009], an approach where distributed computations are programmed using Datalog-based formalisms. In this setting, programs (or queries) are specified over a global schema, and are executed by multiple computing nodes over which the database is distributed. These nodes can perform local computations, and also communicate asynchronously with each other via messages. The model assumes that messages can never be lost but can be arbitrarily delayed. An intrinsic source of inefficiency in such systems are the global barriers raised by the need for synchronization in computing the result of queries. This has motivated a fruitful line of research for isolating classes of queries that can be evaluated in a coordination-free manner [Zinn *et al.*, 2012; Ameloot *et al.*, 2013; 2014; 2015; Alvaro *et al.*, 2014].

It is quite natural to ask wether the results on coordination-free evaluation for declarative database query languages can be transferred to ontology-mediated queries. Undoubtedly, a positive answer to this question, apart from possible applications to declarative networking, will significantly contribute towards more efficient procedures for OBDA, since it will enable distributed ontology-mediated query evaluation in a coordination-free way. The goal of the present work is to initiate the study of coordination-free evaluation for ontology-mediated queries, and give strong indications that the answer to the above challenging question is affirmative.

**Distribution Over Components.** More concretely, we focus on the question whether the answer to an ontology-mediated query can be computed by parallelizing it over the (maximally connected) components of the database, i.e., whether the query *distributes over components*. In other words, given an ontology-mediated query $Q$, the question is whether for every database $D$ the answer to $Q$ over $D$, denoted $Q(D)$, coincides with $\bigcup_{1 \leq i \leq n} Q(D_i)$, where $D_1, \ldots, D_n$ are the components of $D$. In this case, $Q(D)$ can be computed without any communication over a network using a distribution where every computing node is assigned some of the connected components of the database, and every component is assigned to at least one computing node.

The notion of distribution over components has been introduced in [Ameloot *et al.*, 2014], and explicitly considered for Datalog queries in [Ameloot *et al.*, 2015]. It has been shown that *connected* Datalog, that is, the fragment of Datalog where all rule-bodies are connected, provides an effective syntax for Datalog queries that distribute over components, while the problem of deciding whether a Datalog query distributes over components is undecidable.

**Aims and Objectives.** As said, this work is concerned about ontology-mediated queries and their distribution over components. We focus on ontology-mediated queries where the database query is a conjunctive query and the ontology is for-

mulated using existential rules (a.k.a. tuple-generating dependencies or Datalog$^{\pm}$ rules). It is widely accepted that existential rules form a natural and convenient way for modeling ontologies. This has led to a flurry of activity for designing ontology languages based on existential rules; see e.g., [Baget *et al.*, 2011; Calì *et al.*, 2012a; 2012b; Thomazo *et al.*, 2012; Leone *et al.*, 2012]. We consider several ontology-mediated query languages, where the ontology is modeled using either arbitrary existential rules (without syntactic restrictions), or one of the basic decidable classes of existential rules, namely guarded, linear, or sticky (more details are given is Section 2), and we perform similar analysis as the one in [Ameloot *et al.*, 2015] for Datalog queries. In particular, we would like to understand whether connectedness is the right notion for characterizing the fragment of the query languages in question that distributes over components, and understand the complexity of checking distribution over components.

**Our Results.** Our results can be summarized as follows:

▶ We show that for all the ontology-mediated query languages $\mathcal{Q}$ in question, the fragment of $\mathcal{Q}$ that distributes over components has the same expressive power as the fragment of $\mathcal{Q}$ where both the existential rules and the conjunctive query are connected (Theorem 2). For the language where the ontology is expressed using arbitrary existential rules, similar ideas as the ones in [Ameloot *et al.*, 2015] for Datalog queries can be applied. However, the situation changes once we focus on the languages based on guarded, linear, or sticky existential rules, since these languages are not powerful enough to express transitivity axioms, which can be used to compute the connected components of the input database.

▶ The problem of deciding distribution over components is undecidable for all the ontology-mediated query languages that capture Datalog; implicit in [Ameloot *et al.*, 2015]. Interestingly, we show that for the languages based on guarded, linear, or sticky existential rules, the problem is decidable (Theorems 15, 16 and 18).[1] Furthermore, for the languages based on linear and sticky rules, we obtain precise complexity results that range from $\Pi_2^P$ up to coNEXPTIME.

▶ It turned out that our techniques and results have interesting consequences to ontology-mediated query languages based on Description Logics (Corollary 8, and Theorem 19), and to central database query languages, i.e., (unions of) conjunctive queries, and non-recursive Datalog queries (Corollary 11, and Theorems 20 and 21).

## 2 Preliminaries

**Instances and Queries.** Let $\mathbf{C}$, $\mathbf{N}$, and $\mathbf{V}$ be pairwise disjoint countably infinite sets of *constants*, (labeled) *nulls* and (regular) *variables* (used in queries and dependencies), respectively. A *schema* $\mathbf{S}$ is a finite set of relation symbols (or predicates) with associated arity. We write $R/n$ to denote that $R$ has arity $n$. A *term* is either a constant, null or variable. An *atom* over $\mathbf{S}$ is an expression $R(\bar{t})$, where $R$ is a relation symbol in $\mathbf{S}$ of arity $n > 0$ and $\bar{t}$ is an $n$-tuple of terms. A *fact* is an atom whose arguments consist only of constants. An *instance* over $\mathbf{S}$ is a (possibly infinite) set of atoms over $\mathbf{S}$ that contain constants and nulls, while a *database* over $\mathbf{S}$ is a finite set of facts over $\mathbf{S}$. The *active domain* of an instance $I$, denoted $adom(I)$, is the set of all terms occurring in $I$.

A *query* over $\mathbf{S}$ is a mapping $q$ that maps every database $D$ over $\mathbf{S}$ to a set of *answers* $q(D) \subseteq adom(D)^n$, where $n \geq 0$ is the *arity* of $q$. If $n = 0$, then $q$ is a *Boolean query*, and we write $q(D) = 1$ if $() \in q(D)$, and $q(D) = 0$ otherwise.

The usual way of specifying queries is by means of (fragments of) first-order logic. A *conjunctive query* (CQ) $q$ over $\mathbf{S}$ is a conjunction of atoms of the form $\exists \bar{y}\, \phi(\bar{x}, \bar{y})$, where $\bar{x} \cup \bar{y}$ are variables of $\mathbf{V}$, that uses only predicates from $\mathbf{S}$. The free variables of a CQ are called *answer variables*. The evaluation of CQs over instances is defined in terms of *homomorphisms*; we assume the reader is familiar with the standard notion of homomorphism. Let $I$ be an instance over $\mathbf{S}$. The *evaluation* of $q$ over $I$, denoted $q(I)$, is the set of all tuples $h(\bar{x})$ of constants such that $h$ is a homomorphism from $q$ to $I$. Each schema $\mathbf{S}$ and CQ $q = \exists \bar{y}\, \phi(x_1, \ldots, x_n, \bar{y})$ give rise to the $n$-ary query $q_{\phi,\mathbf{S}}$ defined by setting, for every database $D$ over $\mathbf{S}$, $q_{\phi,\mathbf{S}}(D) = \{\bar{c} \in adom(D)^n \mid \bar{c} \in q(D)\}$. Let $\mathsf{CQ}$ be the class of all queries definable by some CQ. Analogously, we define $\mathsf{UCQ}$ as the class of all queries definable by some *union of conjunctive queries* (UCQ), that is, a disjunction of CQs with the same answer variables.

Two queries $q_1$ and $q_2$ over $\mathbf{S}$ are *equivalent*, written $q_1 \equiv q_2$, if, for every database $D$ over $\mathbf{S}$, $q_1(D) = q_2(D)$. A query language $\mathcal{Q}_2$ is *at least as expressive as* query language $\mathcal{Q}_1$, written $\mathcal{Q}_1 \preceq \mathcal{Q}_2$, if, for every $q_1 \in \mathcal{Q}_1$ over schema $\mathbf{S}$, there is $q_2 \in \mathcal{Q}_2$ over $\mathbf{S}$ such that $q_1 \equiv q_2$. $\mathcal{Q}_1$ and $\mathcal{Q}_2$ *have the same expressive power*, written $\mathcal{Q}_1 = \mathcal{Q}_2$, if $\mathcal{Q}_1 \preceq \mathcal{Q}_2 \preceq \mathcal{Q}_1$.

**Tgds for Specifying Ontologies.** An ontology language is a fragment of first-order logic. Here, we focus on languages that are based on tuple-generating dependencies. A *tuple-generating dependency* (tgd) is a first-order sentence of the form $\forall \bar{x} \forall \bar{y} \big(\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})\big)$, where both $\phi$ and $\psi$ are conjunctions of atoms without nulls and constants. For simplicity, we write this tgd as $\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})$, and use comma instead of $\wedge$ for conjoining atoms. We call $\phi$ and $\psi$ the *body* and *head* of the tgd, respectively, and write $sch(\Sigma)$ for the set of predicates occurring in $\Sigma$. An instance $I$ *satisfies* the above tgd if the following holds: For every homomorphism $h$ such that $h(\phi(\bar{x}, \bar{y})) \subseteq I$, there is a homomorphism $h'$ that extends $h$ such that $h'(\psi(\bar{x}, \bar{z})) \subseteq I$. $I$ satisfies a set $\Sigma$ of tgds, denoted $I \models \Sigma$, if $I$ satisfies every tgd in $\Sigma$. Let $\mathsf{TGD}$ be the class of all (finite) sets of tgds.

**Ontology-Mediated Queries.** An *ontology-mediated query* over $\mathbf{S}$ is a triple $(\mathbf{S}, \Sigma, q)$, where $\Sigma \in \mathsf{TGD}$, $q \in \mathsf{CQ}$, and $q$ is over $\mathbf{S} \cup sch(\Sigma)$. We call $\mathbf{S}$ the *data schema*. In general, ontology-mediated queries are defined for arbitrary ontology languages (not only $\mathsf{TGD}$), and arbitrary query languages (not only $\mathsf{CQ}$). In this work, we focus on tgd-based ontology languages and queries definable via CQs. $\mathbf{S}$ is explicitly mentioned in the specification of the ontology-mediated query to highlight that it is interpreted as a query over $\mathbf{S}$. The semantics of an ontology-mediated query is given in terms of *certain answers*. Let $(\mathbf{S}, \Sigma, q)$ be an ontology-mediated query, where $n$ is the arity of $q$. The *certain answers* to $q$ w.r.t. a

[1] For guarded-based queries, this holds under the assumption that the conjunctive query is answer-guarded, i.e., it has an atom that contains all the answer variables.

database $D$ over $\mathbf{S}$ and $\Sigma$ is the set of tuples $cert_{q,\Sigma}(D) = \{\bar{c} \in adom(D)^n \mid \bar{c} \in q(I), \forall I \text{ s.t. } I \supseteq D \text{ and } I \models \Sigma\}$. Recall that $cert_{q,\Sigma}(D)$ coincides with the evaluation of $q$ over the *canonical* instance of $D$ and $\Sigma$ that can be constructed by the *chase*; see e.g., [Calì *et al.*, 2012b]. The chase adds new atoms to $D$ as dictated by $\Sigma$ until the final result, written $chase(D, \Sigma)$, satisfies $\Sigma$. When an existentially quantified variable must be satisfied the chase invents a new null.

**Ontology-Mediated Query Languages.** Every query $Q = (\mathbf{S}, \Sigma, q)$ can be semantically interpreted as a query $q_Q$ over $\mathbf{S}$ by setting $q_Q(D) = cert_{q,\Sigma}(D)$, for every $\mathbf{S}$-database $D$.[2] Thus, we obtain a new query language, denoted $(\mathsf{TGD}, \mathsf{CQ})$, defined as the class of queries $q_Q$, where $Q$ is an ontology-mediated query. However, $(\mathsf{TGD}, \mathsf{CQ})$ is undecidable since, given a database $D$ over $\mathbf{S}$, $\Sigma \in \mathsf{TGD}$, an $n$-ary query $q \in \mathsf{CQ}$ over $\mathbf{S} \cup sch(\Sigma)$, and a tuple $\bar{c} \in \mathbf{C}^n$, the problem of deciding if $\bar{c} \in cert_{q,\Sigma}(D)$ is undecidable [Beeri and Vardi, 1981]. This has led to a flurry of activity for identifying decidable syntactic restrictions. Such a restriction defines a subclass $\mathcal{C}$ of tgds, i.e., $\mathcal{C} \subseteq \mathsf{TGD}$, which in turn gives rise to the decidable query language $(\mathcal{C}, \mathsf{CQ})$. The two main paradigms for ontological reasoning are *guardedness* and *stickiness*:

*Guardedness:* A tgd is *guarded* if its body contains an atom, called *guard*, that contains all the body-variables [Calì *et al.*, 2013]. Let $\mathsf{G}$ be the class of sets of guarded tgds. A key subclass of guarded tgds is the class of *linear* tgds, that is, tgds whose body consists of a single atom [Calì *et al.*, 2012a]. Let $\mathsf{L}$ be the class of sets of linear tgds.

*Stickiness:* The goal of stickiness is to express joins that are not expressible via guarded tgds. The key property underlying this condition is as follows: During the chase, terms that are associated with variables that appear more than once in the body of a tgd (i.e., join variables) are always propagated (or "stick") to the inferred atoms. Due to lack of space we omit the definition that can be found in [Calì *et al.*, 2012b]. Let $\mathsf{S}$ be the class of sticky sets of tgds.

*Weak Versions:* Both $\mathsf{G}$ and $\mathsf{S}$ have a weak version: *Weakly-guarded* [Calì *et al.*, 2013] and *weakly-sticky* [Calì *et al.*, 2012b], respectively. These are highly expressive classes obtained by relaxing guardedness and stickiness so that only those positions that receive nulls during the chase are taken into account. We write $\mathsf{WG}$ and $\mathsf{WS}$ for the class of weakly-guarded and weakly-sticky sets of tgds, respectively.

We refer to the query languages defined above as *ontology-mediated query languages*.

## 3 Distribution of Queries and Connectedness

Our goal is to syntactically characterize the expressive power of the fragment of the query languages in question that guarantees distribution over components. Let us recall the notion of distribution over components [Ameloot *et al.*, 2014]. A set of atoms $A$ is *connected* if for all $a, b \in adom(A)$, there exists a sequence $\alpha_1, \ldots, \alpha_n$ of atoms in $A$ such that $a \in adom(\alpha_1)$, $b \in adom(\alpha_n)$, and $adom(\alpha_i) \cap adom(\alpha_{i+1}) \neq \varnothing$, for each $i \in [n-1]$.[3] We call $B \subseteq A$ a *component* of $A$ if (i) $B$ is connected, and (ii) for every $\alpha \in A \setminus B$, $B \cup \{\alpha\}$ is not connected.[4] Let $co(A)$ be the set of components of $A$.

**Definition 1** *A query $q$ over $\mathbf{S}$* distributes over components *if $q(D) = \bigcup_{D' \in co(D)} q(D')$, for every database $D$ over $\mathbf{S}$.*

Roughly, the centralized answer to $q$ w.r.t. $D$ is precisely obtained when we parallelize $q$ over the components of $D$. Let $\mathsf{DIST}$ be the class of queries that distribute over components. We proceed to characterize the expressive power of the fragment of $(\mathcal{C}, \mathsf{CQ})$, where $\mathcal{C}$ is one of the classes of tgds introduced above. This is done via connectedness. A tgd is connected if the set of atoms occurring in its body is connected, while a set $\Sigma$ of tgds is called connected if every tgd in $\Sigma$ is connected. Given a class $\mathcal{C}$ of tgds, we write $\mathsf{con}\mathcal{C}$ for the class of all (finite) sets of tgds that are connected and fall in $\mathcal{C}$. Similarly, we write $\mathsf{conCQ}$ for the class of all queries definable by some CQ that is connected.

The main result of this section states that every query that is expressible by one of the ontology-mediated query languages $\mathcal{Q}$ in question and distributes over components is equivalent to a connected query that falls in $\mathcal{Q}$, and vice versa. Formally:

**Theorem 2** *For $\mathcal{C} \in \{\mathsf{TGD}, \mathsf{WG}, \mathsf{G}, \mathsf{L}, \mathsf{WS}, \mathsf{S}\}$, $(\mathcal{C}, \mathsf{CQ}) \cap \mathsf{DIST} = (\mathsf{con}\mathcal{C}, \mathsf{conCQ})$.*

Notice that guarded and linear tgds are, by definition, connected. This implies that, for $\mathcal{C} \in \{\mathsf{G}, \mathsf{L}\}$, the above theorem can be equivalently stated as $(\mathcal{C}, \mathsf{CQ}) \cap \mathsf{DIST} = (\mathcal{C}, \mathsf{conCQ})$. We present Theorem 2 in this way for the sake of uniformity. Let us now discuss the key ideas underlying Theorem 2.

For the $(\succeq)$ direction we show, no matter which ontology-mediated query language we consider, that connectedness ensures distribution over components. This is a consequence of the fact that, given a query $Q = (\mathbf{S}, \Sigma, q) \in (\mathsf{con}\mathcal{C}, \mathsf{conCQ})$, for every database $D$ over $\mathbf{S}$ with $co(D) = \{D_1, \ldots, D_m\}$, $chase(D, \Sigma)$ can be partitioned into $\{I_1, \ldots, I_m\}$ such that, for each $i \in [m]$, the atoms of $I_i$ depend only on $D_i$. Thus:

**Proposition 3** *For each $\mathcal{C} \subseteq \mathsf{TGD}$, $(\mathsf{con}\mathcal{C}, \mathsf{conCQ}) \preceq (\mathcal{C}, \mathsf{CQ}) \cap \mathsf{DIST}$.*

We proceed with the $(\preceq)$ direction. It turned out that there is no single reduction (at least no obvious one) from queries that distribute over components to connected queries that is generic enough to deal with all the query languages in question. Nevertheless, our languages can be classified into three groups according to certain properties that are useful for our purposes, and then treat each group separately. These groups and the underlying properties are as follows:

(A) $(\mathsf{TGD}, \mathsf{CQ})$, $(\mathsf{WG}, \mathsf{CQ})$ and $(\mathsf{WS}, \mathsf{CQ})$; these languages are powerful enough to express (a limited form of) transitivity and cartesian products.

(B) $(\mathsf{G}, \mathsf{CQ})$, $(\mathsf{L}, \mathsf{CQ})$; guarded and linear tgds are, by definition, connected.

(C) $(\mathsf{S}, \mathsf{CQ})$; $\mathsf{UCQ}$ is at least as expressive as $(\mathsf{S}, \mathsf{CQ})$, and $(\mathsf{conS}, \mathsf{conCQ})$ is at least as expressive as $\mathsf{conUCQ}$.[5]

[2] For brevity, we use $Q$ both for $(\mathbf{S}, \Sigma, q)$ and the mapping $q_Q$.

[3] Henceforth, we write $[k]$ for $\{1, \ldots, k\}$.

[4] For technical clarity, the notion of component is defined only for sets of atoms that do not contain nullary atoms.

[5] We write $\mathsf{conUCQ}$ for the class of all queries definable by a union of connected CQs.

We consider each one of the above groups, and show that distribution over components implies (semantic) connectedness.

**Group A of Query Languages**

Our main technical tool is the so-called *connecting operator*, based on a construction given in [Ameloot *et al.*, 2015] for showing that distribution over components implies connectedness when we focus on Datalog queries. By applying the connecting operator to a query $Q = (\mathbf{S}, \Sigma, q) \in (\mathsf{TGD}, \mathsf{CQ})$, we obtain the query $Q^{\mathsf{c}} = (\mathbf{S}, \mathsf{c}(\Sigma), \mathsf{c}(q))$ defined as follows. The set $\mathsf{c}(\Sigma)$ consists of the tgds:

$$R(x_1, \ldots, x_n) \rightarrow \mathrm{con}(x_i, x_j),$$

for each $R/n \in \mathbf{S}$ and $i, j \in [n]$, where con is a new binary predicate not in $\Sigma$, and

$$\begin{array}{rcl} \mathrm{con}(x, y) & \rightarrow & \mathrm{con}(y, x) \\ \mathrm{con}(x, y), \mathrm{con}(y, z) & \rightarrow & \mathrm{con}(x, z), \end{array}$$

stating that con is symmetric and transitive; $\mathrm{con}(a, b)$ states that the constants $a$ and $b$ belong to the same component of the input database. In addition, we have the tgd

$$\mathrm{con}(x_1, v), R(x_1, \ldots, x_n) \rightarrow R^{\star}(v, x_1, \ldots, x_n),$$

where $v \notin \{x_1, \ldots, x_m\}$, that annotates every atom of a certain component of the input database with all the constants in this component. Finally, for each $\sigma \in \Sigma$, we add to $\mathsf{c}(\Sigma)$ the connected version of $\sigma$ obtained by replacing each atom $R(\bar{x})$ in $\sigma$ with $R^{\star}(v_\sigma, \bar{x})$, where $v_\sigma$ is a new variable. Analogously, $\mathsf{c}(q)$ is defined as the connected version of $q$. It can be shown that $Q \in \mathsf{DIST}$ implies $Q \equiv Q^{\mathsf{c}}$. Then:

**Proposition 4** *For each $\mathcal{C} \subseteq \mathsf{TGD}$ that is closed under connecting,*[6] $(\mathcal{C}, \mathsf{CQ}) \cap \mathsf{DIST} \preceq (\mathsf{con}\mathcal{C}, \mathsf{conCQ})$.

Interestingly, TGD, WG, and WS are closed under connecting. Hence, Proposition 4 implies that distribution over components implies connectedness for the languages of Group A.

**Group B of Query Languages**

It is clear that G and L are not closed under connecting, and therefore we need to follow a different approach. To this end, we exploit the fact that guarded and linear tgds are, by definition, connected. In particular, we establish a general result for subclasses of conTGD:

**Proposition 5** *For each $\mathcal{C} \subseteq \mathsf{conTGD}$, $(\mathcal{C}, \mathsf{CQ}) \cap \mathsf{DIST} \preceq (\mathcal{C}, \mathsf{conCQ})$.*

It is clear that the above result immediately shows that distribution over components implies connectedness for the languages of Group B. We proceed to explain the key ideas underlying Proposition 5. Fix an arbitrary class $\mathcal{C} \subseteq \mathsf{conTGD}$, and a query $Q = (\mathbf{S}, \Sigma, q) \in (\mathcal{C}, \mathsf{CQ}) \cap \mathsf{DIST}$, where $q/n$ is defined by the CQ $\exists \bar{y}\, \phi(\bar{x}, \bar{y})$. Observe that if $Q$ is unsatisfiable, i.e., there is no database $D$ over $\mathbf{S}$ such that $Q(D) \neq \varnothing$, then the claim holds trivially; simply choose an arbitrary unsatisfiable query from $(\mathcal{C}, \mathsf{conTGD})$. The interesting case is when $Q$ is satisfiable. Let $\{\phi_1, \ldots, \phi_k\}$ be the components of $q$. We assume that $k \geq 2$ since for $k = 1$ the claim holds trivially. Our goal is to show that there exists $i \in [k]$ such that $Q_i = (\mathbf{S}, \Sigma, q_i)$, where $q_i$ is the CQ obtained from $q$ by keeping only the component $\phi_i$, is equivalent to $Q$. Since $Q_i \in (\mathcal{C}, \mathsf{conCQ})$ the claim follows. We proceed by distinguishing the two cases where $q$ is either non-Boolean (i.e., $n > 0$) or Boolean (i.e., $n = 0$).

*Case 1 - Non-Boolean:* Interestingly, we can show that all the answer variables of $q$ occur in a single component. In fact, if an answer variable occurs in more than one component of $q$, then $Q$ is unsatisfiable, which contradicts the fact that $Q$ is satisfiable; this exploits the fact that $\Sigma$ is connected. Therefore, we can refer to *the* component $q_{\bar{x}}$ of $q$ that contains the answer variables of $q$. Notably, the query obtained from $Q$ by replacing $q$ with $q_{\bar{x}}$ is equivalent to $Q$:

**Lemma 6** *$Q \equiv Q_{\bar{x}}$, where $Q_{\bar{x}} = (\mathbf{S}, \Sigma, q_{\bar{x}})$.*

It is clear that $Q_{\bar{x}} \in (\mathcal{C}, \mathsf{conCQ})$, and the claim when $q$ is non-Boolean follows by Lemma 6.

*Case 2 - Boolean:* Let us now consider the Boolean case. Although we cannot refer to the component $q_{\bar{x}}$ of $q$, we can show that there exists a component that gives rise to a query equivalent to $Q$. Given a query $\hat{Q} = (\hat{\mathbf{S}}, \hat{\Sigma}, \hat{q}) \in (\mathcal{C}, \mathsf{CQ}) \cap \mathsf{DIST}$, where $\hat{q}$ is Boolean and $\psi_1, \ldots, \psi_m$ are its components, let $\hat{Q}_i$ be the query obtained from $\hat{Q}$ by keeping only the component $\psi_i$ of $\hat{q}$, and $\hat{Q}_i^-$ the query obtained from $\hat{Q}$ by removing the component $\psi_i$. We can show that, for each $i \in [m]$, $\hat{Q}_i \subseteq \hat{Q}_i^-$ or $\hat{Q}_i^- \subseteq \hat{Q}_i$.[7] Thus, starting from $Q$, and repeatedly applying this result, we will find an integer $i \in [k]$ such that $Q_i \subseteq Q_i^-$. It is not difficult to show that $Q \equiv Q_i$. Thus:

**Lemma 7** *There exists $i \in [k]$ such that $Q \equiv Q_i$, where $Q_i = (\mathbf{S}, \Sigma, q_i)$.*

It is clear that the query $Q_i$ provided by Lemma 7 belongs to $(\mathcal{C}, \mathsf{conCQ})$, and the claim when $q$ is Boolean follows.

**Description Logics.** Before we proceed to the third group of query languages, we would like to briefly discuss an interesting consequence of Proposition 5 to ontology-mediated query languages based on Description Logics (DLs). As for classes of tgds, a DL $\mathcal{L}$ gives rise to the query language $(\mathcal{L}, \mathsf{CQ})$. One of the most prominent and well-studied DLs is $\mathcal{EL}$ [Baader, 2003]. It is known that an $\mathcal{EL}$-TBox can be rewritten as a set of guarded tgds [Calì *et al.*, 2012a]. This fact, together with Propositions 3 and 5, allow us to show that:

**Corollary 8** $(\mathcal{EL}, \mathsf{CQ}) \cap \mathsf{DIST} = (\mathcal{EL}, \mathsf{conCQ})$.

**Group C of Query Languages**

S is not closed under connecting, and $\mathsf{S} \not\subseteq \mathsf{conTGD}$. Thus, we need to apply different techniques for establishing the desired result. Our main technical tool is as follows:

**Proposition 9** *For each $\mathcal{C} \subseteq \mathsf{TGD}$ such that*

1. $(\mathcal{C}, \mathsf{CQ}) \preceq \mathsf{UCQ}$;
2. $\mathsf{conUCQ} \preceq (\mathsf{con}\mathcal{C}, \mathsf{conCQ})$,

*it holds that $(\mathcal{C}, \mathsf{CQ}) \cap \mathsf{DIST} \preceq (\mathsf{con}\mathcal{C}, \mathsf{conCQ})$.*

[6] This means that, for each set $\Sigma \in \mathcal{C}$, $\mathsf{c}(\Sigma) \in \mathcal{C}$.

[7] As usual, $Q_1 \subseteq Q_2$ means that for every source database $D$, $Q_1(D) \subseteq Q_2(D)$.

To establish Proposition 9, it suffices to show the following lemma, which can be done by exploiting similar ideas as the ones developed above for proving Proposition 5:

**Lemma 10** $\mathsf{UCQ} \cap \mathsf{DIST} \preceq \mathsf{conUCQ}$.

Having the above proposition in place, to establish the desired result for $(\mathsf{S}, \mathsf{CQ})$ it remains to show that $(\mathsf{S}, \mathsf{CQ}) \preceq \mathsf{UCQ}$ and $\mathsf{conUCQ} \preceq (\mathsf{conS}, \mathsf{conCQ})$. The former is implicit in [Gottlob *et al.*, 2014], where it is shown that evaluation for $(\mathsf{S}, \mathsf{CQ})$ is UCQ-rewritable. The latter is shown by exploiting the fact that every CQ of the form $\exists \bar{y}\, \phi(\bar{x}, \bar{y})$ can be converted into the sticky tgd $\phi(\bar{x}, \bar{y}) \rightarrow ans(\bar{x}, \bar{y})$; this tgd is sticky since all the body-variables are propagated to the head.

**Database Query Languages.** We conclude this section by discussing some interesting consequences to central database query languages, in particular CQ, UCQ, and NRDat.[8] First, notice that Proposition 3 can be established for CQ and UCQ. This fact, together with Lemma 10, which holds also for CQ, implies that $\mathsf{CQ} \cap \mathsf{DIST}$ ($\mathsf{UCQ} \cap \mathsf{DIST}$) has the same expressive power as conCQ (conUCQ). Regarding NRDat, we can show that the class of tgds that corresponds to non-recursive Datalog programs enjoys the two properties stated in Proposition 9. Hence, Propositions 3 and 9 imply that:

**Corollary 11** *For each language* $\mathcal{Q} \in \{\mathsf{CQ}, \mathsf{UCQ}, \mathsf{NRDat}\}$, $\mathcal{Q} \cap \mathsf{DIST} = \mathsf{con}\mathcal{Q}$.

## 4 Deciding Distribution Over Components

The question that comes up is whether distribution over components for the query languages considered so far can be decided. This problem, dubbed $\mathsf{Dist}(\mathcal{Q})$ with $\mathcal{Q}$ being a query language, accepts as input a query $Q \in \mathcal{Q}$, and asks whether $Q \in \mathsf{DIST}$. Notice that Theorem 2 only says that $\mathsf{Dist}(\mathcal{Q})$ is equivalent to the problem of checking whether a query is semantically (not syntactically) connected, and thus it does not provide a decision procedure for our problem.

We start our analysis be recalling a recent negative result: $\mathsf{Dist}(\mathsf{Dat})$ is undecidable [Ameloot *et al.*, 2015], where Dat is the class of all queries definable by some Datalog query. Thus, for every $\mathcal{C} \subseteq \mathsf{TGD}$ that captures Datalog programs, we immediately obtain that $\mathsf{Dist}(\mathcal{C}, \mathsf{CQ})$ is undecidable.[9] TGD, WG and WS are such classes, and hence:

**Theorem 12** *For* $\mathcal{Q} \in \{(\mathsf{TGD}, \mathsf{CQ}), (\mathsf{WG}, \mathsf{CQ}), (\mathsf{WS}, \mathsf{CQ})\}$, $\mathsf{Dist}(\mathcal{Q})$ *is undecidable.*

This negative result rules out weakly-guarded and weakly-sticky sets of tgds. What about their non-weak versions? Do they ensure the decidability of Dist? This will be the subject of the remainder of this section.

**Distribution and Guardedness**

We show that our problem is decidable if we focus on the answer-guarded fragment of $(\mathsf{G}, \mathsf{CQ})$. A CQ $q$ is *answer-guarded* if it includes an atom that contains all the answer variables [Bárány *et al.*, 2013]. Let AGCQ be the class of all queries definable by some answer-guarded CQ. Notice that AGCQ is a broad class of queries, which includes all Boolean CQs. Our goal is to show that $\mathsf{Dist}(\mathsf{G}, \mathsf{AGCQ})$ is decidable. Fix a query $Q = (\mathbf{S}, \Sigma, q) \in (\mathsf{G}, \mathsf{AGCQ})$, where $\{q_1, \ldots, q_k\}$ are the components of $q$. Recall that $Q_i$ is the query obtained from $Q$ by keeping only the component $q_i$ of $q$. By exploiting Proposition 3, and Lemmas 6 and 7, we can show that:

**Lemma 13** $Q \in \mathsf{DIST}$ *iff one of the following holds:*

1. $Q$ *is unsatisfiable;*
2. *There exists* $i \in [k]$ *such that* $Q_i \in (\mathsf{G}, \mathsf{AGCQ})$ *and* $Q_i \subseteq Q$.

Checking whether $Q$ is unsatisfiable can be reduced to the containment problem for $(\mathsf{G}, \mathsf{AGCQ})$. Thus, Lemma 13 provides an algorithm for $\mathsf{Dist}(\mathsf{G}, \mathsf{AGCQ})$ under the assumption that $\mathsf{Cont}(\mathsf{G}, \mathsf{AGCQ})$ is decidable.[10] It remains to show that:

**Proposition 14** $\mathsf{Cont}(\mathsf{G}, \mathsf{AGCQ})$ *is decidable.*

The above result is shown by a reduction to $\mathsf{Cont}(\mathsf{GDat})$, where GDat is the class of queries definable by some *guarded Datalog* query [Bárány *et al.*, 2012]. Guarded Datalog requires an atom with an extensional predicate to contain all the variables in the head. $\mathsf{Cont}(\mathsf{GDat})$ is in 2ExpTime as it can be reduced to the satisfiability problem for guarded negation fixed point logic [Bárány *et al.*, 2015]. Our reduction exploits a construction given in [Bárány *et al.*, 2013] for rewriting a query of $Q = (\mathbf{S}, \Sigma, q) \in (\mathsf{G}, \mathsf{AGCQ})$ into an equivalent Datalog query $Q_{\mathsf{Dat}} = (\Pi, Ans)$, where each rule of $\Pi$ has a guard atom (possibly with an intensional predicate) that contains all the head-variables. This construction assumes that $q$ is answer-guarded; this is the reason why we focus on answer-guarded queries. From the above discussion, we get that:

**Theorem 15** $\mathsf{Dist}(\mathsf{G}, \mathsf{AGCQ})$ *is decidable.*

The decision procedure underlying Theorem 15 shows that $\mathsf{Dist}(\mathsf{G}, \mathsf{AGCQ})$ is elementary. However, its exact complexity is unknown. Moreover, the decidability of $\mathsf{Dist}(\mathsf{G}, \mathsf{CQ})$, i.e., when the CQ is not answer-guarded, is still open.

**Distribution and Linearity**

It is easy to verify that the proof of Theorem 15 applies even if we focus on linear tgds; thus, $\mathsf{Dist}(\mathsf{L}, \mathsf{AGCQ})$ is decidable. Interestingly, by exploiting the fact that evaluation for $(\mathsf{L}, \mathsf{CQ})$ is UCQ-rewritable [Gottlob *et al.*, 2014], we can significantly strengthen the above result. We can show that:

**Theorem 16** $\mathsf{Dist}(\mathsf{L}, \mathsf{CQ})$ *is* PSpace*-complete, and* $\Pi_2^P$*-complete if the arity of the schema is fixed.*

Fix a query $Q = (\mathbf{S}, \Sigma, q) \in (\mathsf{L}, \mathsf{CQ})$. By applying query rewriting techniques, we can construct $Q_r \in \mathsf{UCQ}$ such that $Q \equiv Q_r$ [Gottlob *et al.*, 2014]. Clearly, $Q \in \mathsf{DIST}$ iff $Q_r \in \mathsf{DIST}$. Hence, it would be quite beneficial to first understand the problem $\mathsf{Dist}(\mathsf{UCQ})$. One may be tempted to think that a UCQ distributes over components iff each of its disjuncts does. It is easy to show that this is not the case; consider, e.g., the UCQ $\exists x \exists y\, (R(x) \wedge S(y)) \vee \exists x\, R(x)$. The reason for this is the fact that some disjuncts that do not distribute over components maybe subsumed by other ones that distribute over components. To formalize this we need an

[8] All queries definable by some non-recursive Datalog query.

[9] For brevity, we write $\mathsf{Dist}(\mathcal{C}, \mathsf{CQ})$ instead of $\mathsf{Dist}((\mathcal{C}, \mathsf{CQ}))$.

[10] For a query language $\mathcal{Q}$, $\mathsf{Cont}(\mathcal{Q})$ accepts as input two queries $Q_1, Q_2 \in \mathcal{Q}$ of the same arity, and asks whether $Q_1 \subseteq Q_2$.

auxiliary notation. For a CQ $q(\bar{x}) = \exists \bar{y}\,(\bigwedge_{i=1}^{n} R_i(\bar{x}, \bar{y}))$, let $D[q]$ the database $\{R_i(c(\bar{x}), c(\bar{y}))\}_{i\in[n]}$ obtained after replacing each variable $v$ in $q$ with a fresh constant $c(v)$. Then:

**Lemma 17** *For $Q(\bar{x}) \in \mathsf{UCQ}$, $Q \in \mathsf{DIST}$ iff for each $q \in Q$, there exists $D' \in co(D[q])$ such that $c(\bar{x}) \in Q(D')$.*

Let $Q = (\mathbf{S}, \Sigma, q(\bar{x})) \in (\mathsf{L}, \mathsf{CQ})$, and let $Q_r(\bar{x}) \in \mathsf{UCQ}$ be the rewriting obtained by applying the rewriting algorithm in [Gottlob *et al.*, 2014]; recall that $Q \equiv Q_r$. This algorithm constructs $Q_r$ starting from $q$ and exhaustively applying two steps: *rewriting* and *minimization*. The rewriting step resolves an atom in the disjunct $q'$ under consideration using a linear tgd of $\Sigma$, while the minimization step is responsible for unifying some atoms of $q'$. Lemma 17 implies that $Q \notin \mathsf{DIST}$ iff there exists $q' \in Q_r$ such that, for every $D' \in co(D[q'])$, $c(\bar{x}) \in Q(D')$. This suggests the following decision procedure for the complement of $\mathsf{Dist}(\mathsf{L}, \mathsf{CQ})$:

1. Construct a CQ $q'(\bar{x}) \in Q_r$ by nondeterministically applying rewriting and minimization steps;
2. For each $D' \in co(D)$, if $c(\bar{x}) \in Q(D')$, then $Reject$;
3. $Accept$.

Each step of the above procedure uses polynomial space. This is shown by exploiting the fact that each $q' \in Q_r$ cannot have more atoms than the original CQ $q$ due to the linearity of tgds, and also the fact that evaluation for $(\mathsf{L}, \mathsf{CQ})$ is in PSPACE; the latter is implicit in [Johnson and Klug, 1984]. In case the arity of the schema is fixed, $q'$ can be nondeterministically constructed in polynomial time, while evaluation for $(\mathsf{L}, \mathsf{CQ})$ is in NP; implicit in [Johnson and Klug, 1984]. Thus, the above procedure gives a $\Sigma_2^P$ upper bound. Hence, $\mathsf{Dist}(\mathsf{L}, \mathsf{CQ})$ is in PSPACE, and in $\Pi_2^P$ if the arity of the schema is fixed.

The lower bounds are established by a reduction from a restricted version of $\mathsf{Cont}(\mathsf{L}, \mathsf{CQ})$. We define $\mathsf{RestCont}(\mathcal{C}, \mathsf{CQ})$ to be the problem of checking $Q_1 \subseteq Q_2$, where $Q_1 = (\mathbf{S} \cup \{P\}, \Sigma, q_1)$ and $Q_2 = (\mathbf{S} \cup \{P\}, \Sigma, q_2)$ with $P \notin \mathbf{S}$ being a unary predicate not in $\Sigma$, $q_1$, $q_2$, given: (i) a set $\Sigma \in \mathcal{C}$ of connected tgds, (ii) a Boolean connected CQ $q_1$ with at least one variable $v_{q_1}$ that occurs only in atoms with a predicate of $\mathbf{S}$, and (iii) a Boolean CQ $q_2$ that uses only predicates of $\mathbf{S}$. It holds that $Q_1 \subseteq Q_2$ iff $(\mathbf{S} \cup \{P\}, \Sigma, q_1 \wedge P(v_{q_1}) \wedge q_2) \in \mathsf{DIST}$, which in turn implies that we have a logspace reduction from $\mathsf{RestCont}(\mathsf{L}, \mathsf{CQ})$ to $\mathsf{Dist}(\mathsf{L}, \mathsf{CQ})$. Therefore, to obtain the desired lower bounds for $\mathsf{Dist}(\mathsf{L}, \mathsf{CQ})$, it remains to show that $\mathsf{RestCont}(\mathsf{L}, \mathsf{CQ})$ is PSPACE-hard, and $\Pi_2^P$-hard for fixed arity. The former can be shown by a reduction from query answering under linear tgds, while the latter is implicit in [Bienvenu *et al.*, 2012] (see the proof of Theorem 9). In fact, the $\Pi_2^P$-hardness holds even for tgds of the form $P(x) \rightarrow R(x)$.

**Distribution and Stickiness**

We proceed to show that $\mathsf{Dist}(\mathsf{S}, \mathsf{CQ})$ is decidable, and pinpoint the complexity of the problem:

**Theorem 18** $\mathsf{Dist}(\mathsf{S}, \mathsf{CQ})$ *is* coNEXPTIME*-complete,*[11] *and $\Pi_2^P$-complete if the arity of the schema is fixed.*

Evaluation for $(\mathsf{S}, \mathsf{CQ})$ is UCQ-rewritable [Gottlob *et al.*, 2014]. This allows us to exploit the algorithm devised above for solving the complement of $\mathsf{Dist}(\mathsf{L}, \mathsf{CQ})$. However, since sticky tgds may have more than one atom in their body, the space we need during the execution of the algorithm is not polynomial anymore; actually, we need exponential space. Nevertheless, it can be shown that for $(\mathsf{S}, \mathsf{CQ})$ this algorithm terminates after exponentially many steps, which implies a NEXPTIME upper bound for the complement of our problem. This exploits the fact that query evaluation for $(\mathsf{S}, \mathsf{CQ})$ is in EXPTIME [Calì *et al.*, 2012b]. In case the arity of the schema is fixed, the same analysis as for $(\mathsf{L}, \mathsf{CQ})$ applies, and thus we obtain a $\Sigma_2^P$ upper bound for the complement of our problem. The fact that evaluation for $(\mathsf{S}, \mathsf{CQ})$ in case of fixed arity is in NP has been shown in [Lukasiewicz *et al.*, 2015].

To show coNEXPTIME-hardness for $\mathsf{Dist}(\mathsf{S}, \mathsf{CQ})$, we first establish the same for $\mathsf{Cont}(\mathsf{conS}, \mathsf{conCQ})$ by giving a reduction from the so-called *Exponential Tiling Problem* [Johnson, 1990]; for technical reasons, we assume databases with at least two constants. The desired coNEXPTIME-hardness is obtained by reducing $\mathsf{Cont}(\mathsf{conS}, \mathsf{conCQ})$ to $\mathsf{Dist}(\mathsf{S}, \mathsf{conCQ})$. The $\Pi_2^P$-hardness in case the arity of the schema is fixed is obtained in the same way as for $(\mathsf{L}, \mathsf{CQ})$.

**Description Logics and Database Query Languages**

Interestingly, by exploiting the ideas developed above for understanding the complexity of Dist, we can show that:

**Theorem 19** $\mathsf{Dist}(\mathcal{EL}, \mathsf{CQ})$ *is* EXPTIME*-complete.*

For the upper bound, we relax Lemma 13 in order to deal with arbitrary (beyond answer-guarded) CQs, and then use the fact that $\mathsf{Cont}(\mathcal{EL}, \mathsf{CQ})$ is in EXPTIME [Bienvenu *et al.*, 2012]. The lower bound exploits the problem $\mathsf{RestCont}^+$ obtained from RestCont dy dropping the third condition in the definition of an instance of RestCont. $\mathsf{RestCont}^+(\mathcal{EL}, \mathsf{CQ})$ is known to be EXPTIME-hard [Bienvenu *et al.*, 2012], which is not the case for $\mathsf{RestCont}(\mathcal{EL}, \mathsf{CQ})$.

We conclude with the complexity of Dist when we focus on central database query languages. We show that:

**Theorem 20** $\mathsf{Dist}(\mathsf{CQ})$ *and* $\mathsf{Dist}(\mathsf{UCQ})$ *are* NP*-complete, even if the arity of the schema is fixed.*

By exploiting Lemma 17, we can design a guess-and-check algorithm for $\mathsf{Dist}(\mathsf{UCQ})$ that runs in polynomial time; the NP-hardness is shown by reduction from $\mathsf{Cont}(\mathsf{CQ})$. Finally, by giving a proof similar to the one for Theorem 18, we show:

**Theorem 21** $\mathsf{Dist}(\mathsf{NRDat})$ *is* coNEXPTIME*-complete, even if the arity of the schema is fixed.*[11]

## 5 Conclusions

Here are some interesting open problems that we are planning to tackle: (i) The complexity of deciding distribution over components for guarded-based queries is still open, (ii) distribution over components in the presence of equality and denial constraints is not well understood, and (iii) we do not know how distribution over components behaves in the case of non-monotonic queries.

**Acknowledgements.** Austrian Science Fund (FWF), projects P25207-N23, Y698 and W1255-N23, and Vienna Science and Technology Fund (WWTF), project ICT12-015.

[11] The lower bound assumes databases with at least two constants.

## References


[Alvaro *et al.*, 2014] Peter Alvaro, Neil Conway, Joseph M. Hellerstein, and David Maier. Blazes: Coordination analysis for distributed programs. In *ICDE*, pages 52–63, 2014.

[Ameloot *et al.*, 2013] Tom J. Ameloot, Frank Neven, and Jan Van den Bussche. Relational transducers for declarative networking. *J. ACM*, 60(2):15, 2013.

[Ameloot *et al.*, 2014] Tom J. Ameloot, Bas Ketsman, Frank Neven, and Daniel Zinn. Weaker forms of monotonicity for declarative networking: A more fine-grained answer to the CALM-conjecture. In *PODS*, pages 64–75, 2014.

[Ameloot *et al.*, 2015] Tom J. Ameloot, Bas Ketsman, Frank Neven, and Daniel Zinn. Datalog queries distributing over components. In *ICDT*, pages 308–323, 2015.

[Baader, 2003] Franz Baader. Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In *IJCAI*, pages 319–324, 2003.

[Baget *et al.*, 2011] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011.

[Bárány *et al.*, 2012] Vince Bárány, Balder ten Cate, and Martin Otto. Queries with guarded negation. *PVLDB*, 5(11):1328–1339, 2012.

[Bárány *et al.*, 2013] Vince Bárány, Michael Benedikt, and Balder ten Cate. Rewriting guarded negation queries. In *MFCS*, pages 98–110, 2013.

[Bárány *et al.*, 2015] Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded negation. *J. ACM*, 62(3):22, 2015.

[Beeri and Vardi, 1981] Catriel Beeri and Moshe Y. Vardi. The implication problem for data dependencies. In *ICALP*, pages 73–85, 1981.

[Bienvenu *et al.*, 2012] Meghyn Bienvenu, Carsten Lutz, and Frank Wolter. Query containment in description logics reconsidered. In *KR*, 2012.

[Bienvenu *et al.*, 2014] Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive datalog, csp, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014.

[Calì *et al.*, 2012a] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.

[Calì *et al.*, 2012b] Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012.

[Calì *et al.*, 2013] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013.

[Gottlob *et al.*, 2014] Georg Gottlob, Giorgio Orsi, and Andreas Pieris. Query rewriting and optimization for ontological databases. *ACM Trans. Database Syst.*, 39(3):25:1–25:46, 2014.

[Johnson and Klug, 1984] David S. Johnson and Anthony C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984.

[Johnson, 1990] David S. Johnson. A catalog of complexity classes. In *Handbook of Theoretical Computer Science*, pages 67–161. 1990.

[Leone *et al.*, 2012] Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri. Efficiently computable Datalog$^{\exists}$ programs. In *KR*, 2012.

[Loo *et al.*, 2009] Boon Thau Loo, Tyson Condie, Minos N. Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. Declarative networking. *Commun. ACM*, 52(11):87–95, 2009.

[Lukasiewicz *et al.*, 2015] Thomas Lukasiewicz, Maria Vanina Martinez, Andreas Pieris, and Gerardo I. Simari. From classical to consistent query answering under existential rules. In *AAAI*, pages 1546–1552, 2015.

[Thomazo *et al.*, 2012] Michaël Thomazo, Jean-François Baget, Marie-Laure Mugnier, and Sebastian Rudolph. A generic querying algorithm for greedy sets of existential rules. In *KR*, 2012.

[Zinn *et al.*, 2012] Daniel Zinn, Todd J. Green, and Bertram Ludäscher. Win-move is coordination-free (sometimes). In *ICDT*, pages 99–113, 2012.